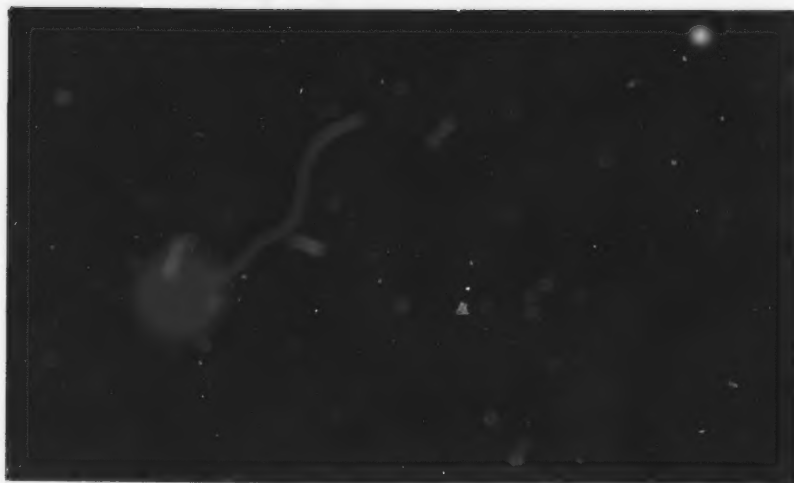




**CENTRE
DE RECHERCHE
SUR LES TRANSPORTS
(CRT)**

**CENTRE
FOR RESEARCH
ON TRANSPORTATION**

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE



C.P. 6128, SUCCURSALE A
MONTRÉAL (QUÉBEC)
CANADA
H3C 3J7

TÉLÉPHONE : (514) 343-7575
TÉLEX : 05-24146 BIBPOLYTEC
TÉLÉCOPIEUR (FAX) : (514) 343-7121

To appear in :
"Publications in Operations Research Series", vol. 9, North-Holland, 1990.

CRT-669

ENHANCEMENTS TO THE CLARKE AND WRIGHT ALGORITHM
VIA GENERALIZATION AND LOOK-AHEAD

by

Jean-Yves Potvin¹

¹ Centre de recherche sur les transports, Université de Montréal, C.P. 6128,
Succursale "A", Montréal, Québec, Canada H3C 3J7.

Centre de recherche sur les transports - Publication #669

December 1989

90-01605/

Dépôt légal - 4e trimestre 1989

Résumé. Dans cet article, nous présentons deux variantes de l'algorithme classique de Clarke et Wright. L'une des variantes généralise l'approche classique en considérant un voisinage plus large de la solution courante. L'autre variante introduit un facteur de pénalité qui atténue le comportement vorace de l'algorithme original. Des résultats expérimentaux montrent que ces deux variantes génèrent de meilleures solutions que l'algorithme classique de Clarke et Wright. En particulier, l'approche fondée sur les facteurs de pénalité génère des solutions qui se comparent avantageusement à celles de l'approche généralisée, et ce, avec une complexité qui demeure dans l'ordre de $N^2 \log N$, où N est le nombre de noeuds dans le réseau.

Mots Clés. Problème du Voyageur de Commerce, Algorithme de Clarke et Wright, Heuristiques.

Abstract. In this paper, we present two variants of the classical Clarke and Wright algorithm. One variant generalizes the savings measure by considering a larger neighborhood of the current solution. The other variant introduces a penalty (or look-ahead) factor aimed at reducing the "myopic" behavior of the classical approach. Computational results show that these two variants generate better solutions than the classical Clarke and Wright algorithm. In particular, the penalty approach generates solutions that compare favorably to the generalized approach, even with a complexity that remains in the order of $N^2 \log N$, where N is the number of nodes in the network.

Key Words. Traveling Salesman Problem, Clarke and Wright Algorithm, Heuristics.

Enhancements to the Clarke and Wright Algorithm via Generalization and Look-Ahead

Jean-Yves Potvin

Centre de Recherche sur les Transports
Université de Montréal

Abstract. In this paper, we present two variants of the classical Clarke and Wright algorithm. One variant generalizes the savings measure by considering a larger neighborhood of the current solution. The other variant introduces a penalty (or look-ahead) factor aimed at reducing the "myopic" behavior of the classical approach. Computational results show that these two variants generate better solutions than the classical Clarke and Wright algorithm. In particular, the penalty approach generates solutions that compare favorably to the generalized approach, even with a complexity that remains in the order of $N^2 \log N$.

Section 1. Introduction

The Clarke and Wright algorithm [1] is known to be a very good heuristic approach for solving Traveling Salesman Problems (TSP). Basically, it uses a measure of saving for generating a tour, and this measure looks quite powerful when spatial considerations are of prime importance (as it is the case when no additional constraints, like capacity or time windows, are introduced in the basic TSP problem).

If the saving measure as such represents a good starting point for building a tour, the way it is used in the Clarke and Wright algorithm can certainly be improved. This algorithm is in fact a greedy heuristic, which selects at each step (in a very myopic way) the best available saving measure, so as to make a step further towards the final solution. In this paper, we show how this approach can be improved via a generalization of the saving measure and via the introduction of a look-ahead factor aimed at reducing the myopic behavior of this algorithm.

Section 2 will first briefly describe the classical Clarke and Wright algorithm and emphasize its basic weaknesses. Section 3 then presents the first modification to the basic algorithm and introduces a look-ahead factor aimed at reducing the algorithm's myopic behavior. Section 4 then describes a second modification to the basic algorithm

and explains how the savings measurements can be generalized in order to consider at each step a larger neighborhood of the current solution. Finally, section 5 shows computational results that compare the performance of the classical Clarke and Wright algorithm with these two variants.

Section 2. The Clarke and Wright algorithm

Let a network $G=(N,L,D)$ be defined with N the set of nodes, L the set of links and $D=[d_{ij}]$ the distance matrix. The TSP problem tries to identify the Hamiltonian cycle in G of minimal length. Due to the difficulty of this problem, which is known to be NP-hard, many heuristic or approximate procedures have been developed, among which the Clarke and Wright algorithm emerged as one of the best. It is also a very "popular" approach which is widely used in various commercial routing packages.

This algorithm works as follows. First, one of the node is selected as the starting point of the tour and is called the depot (in "real-life" problems, the depot is already located on the network and represents a distinct physical entity like a garage for housing transportation vehicles). Initially, each one of the remaining nodes is conceptually serviced by an individual tour starting and ending at the depot. Hence, there are $N-1$ initial tours.

Now, if we assume that the nodes are numbered from 1 to N , and that node 1 is the depot, we can describe this algorithm in the following way:

Step 1. Compute savings $s_{i,j} = d_{1,i} + d_{1,j} - d_{i,j}$ for $i,j=2,3,\dots,N$, $i \neq j$.

Step 2. Order the savings from largest to smallest.

Step 3. Starting at the top of the savings list and moving downwards, form larger subtours by linking the two end points of distinct tours with maximal savings. Repeat until a single tour is formed.

As we can see, the savings are calculated only once for all pairs of nodes and the algorithm selects at each step the best feasible saving (i.e. the maximal saving involving two end points of distinct tours). The calculation of the matrix $S=[s_{i,j}]$ in step 1 requires $O(N^2)$ operations and the savings can be sorted into nonincreasing order via the Heapsort method of Williams in $O(N^2 \lg N)$ operations [4]. Step 3

requires $O(N^2)$ operations since there are that many savings to consider. Thus, the Clarke and Wright savings procedure requires on the order of $(N^2 \lg N)$ computations.

A good way to understand the greedy nature of this algorithm is to transform the original network into a "savings network". This is done by removing the depot and its incident links and by setting the length $d_{i,j}$ of the remaining links to $s_{i,j}$ (see the example in figure 1, where the black square is the depot and the numbers are the lengths of the links).

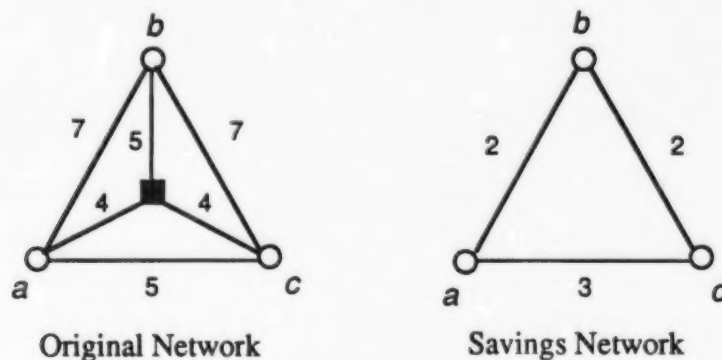


Fig. 1 Original Network vs Savings Network

Based on the savings network, it is easy to see that the Clarke and Wright algorithm is similar to the Kruskal algorithm, which is a very well known greedy procedure for generating maximal (minimal) spanning trees [2]. Here, the Clarke and Wright algorithm tries to identify a maximal spanning chain in the savings network by iteratively selecting at each step the feasible link with maximal length and adding it to the current set of selected links (i.e. to the current solution). A link between two nodes i and j is feasible with respect to the current solution if, by adding this link, a cycle is not created *and* the degree of both i and j does not exceed two (so as to get a chain). The procedure stops when it is not possible to add a new link in the current solution. In such a situation, a spanning chain has been identified. The Clarke and Wright's tour is simply the sequence of stops in the spanning chain, with the addition of the depot on both ends of that sequence.

As it is well known, the Kruskal algorithm is optimal. However, this optimal behavior is not preserved when additional constraints are imposed on the degree of the nodes in the final solution (as it is the case when a chain is built rather than a tree). Hence, the Clarke and Wright algorithm is a heuristic procedure that identifies a spanning chain in the savings network, but not necessarily a maximal one.

Having presented the classical Clarke and Wright procedure, the next section will now describe a variant aimed at reducing its myopic behavior via the introduction of a look-ahead factor in the saving measurement.

Section 3. The Clarke and Wright algorithm with look-ahead

This approach has been proposed by Tillman and Cain in the context of multi-depot problems [3]. Basically, a penalty correction factor is added to the saving measure in the following way:

Suppose that link (i,j) can be selected at a given iteration, with saving $s_{i,j}$. If we decide not to choose that link, then i (j) will be linked with some other node, say u (v), at a later iteration. We can thus evaluate the penalty for not selecting link (i,j) as $p_{i,j} = p_{i,u} + p_{v,j}$ where $p_{i,u} = \max(0, s_{i,j} - s_{i,u})$ and $p_{v,j} = \max(0, s_{i,j} - s_{v,j})$. In general, we can define a penalty $p_{i,j}$ for each feasible link at a given iteration as:

$$p_{i,j} = \min_u \{p_{i,u} \mid \text{for all feasible links } (i,u), u \neq i\} + \min_v \{p_{v,j} \mid \text{for all feasible links } (v,j), v \neq j\}$$

The classical saving measure and the penalty factor are then incorporated into a new modified saving measure $s_{i,j}' = \alpha s_{i,j} + \beta p_{i,j}$ where α and β are positive constants, $\alpha + \beta = 1$. The penalty factor thus tries to overcome the myopic behavior of the classical approach by evaluating what can be lost later if a link is not chosen at a given iteration.

The Clarke and Wright algorithm with penalty must reevaluate the modified savings at each iteration in order to select the maximal one. We must however observe that $N-1$ savings at most must be reevaluated at each step, since the penalty factors are positive for those savings $s_{i,j}$ which are maximal with respect either to node i or node j . Hence, for each node i , the penalty factor is added only to saving $s_{ij}^* = \max_j \{s_{i,j} \mid \text{for all feasible links } (i,j), i \neq j\}$.

Assuming that we maintain $N-1$ saving lists of length $N-2$ (one list for each node), we can describe the penalty algorithm in the following way:

- Step 1.* Compute savings $s_{i,j} = d_{1,i} + d_{1,j} - d_{i,j}$ for $i,j=2,3,\dots,N$, $i \neq j$ and put them in the appropriate lists.
- Step 2.* Order the savings from largest to smallest in each list.
- Step 3.* Starting at the top of each list and moving downwards, evaluate the modified saving measure by adding the penalty factor to the maximal feasible saving in each list. Form a larger subtour by linking the two end points of distinct tours with the maximal modified saving measure. Repeat until a single tour is formed.

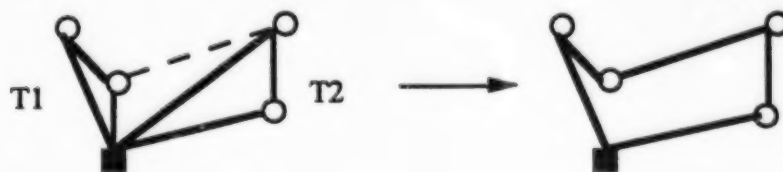
As mentioned in section 2, evaluating and sorting the savings require $O(N^2 \lg N)$ operations. Step 3 requires $O(N^2)$ operations because: (1) there are that many savings to consider (2) Step 3 is repeated $N-2$ times and, at each iteration, at most $N-1$ savings are reevaluated and compared. By adding these two components we get $O(N^2)$ as the complexity of step 3. Hence, the overall complexity of this algorithm is $O(N^2 \lg N)$, which is identical to the complexity of the classical approach.

It must be noticed that the set of savings that is considered at each step by the penalty approach (i.e. the neighborhood of the current solution) is the same as in the case of the classical approach. The penalty approach simply modifies the way the "best" saving is selected in that set. In the next section, we describe a "generalized" Clarke and Wright algorithm where this set is enlarged so as to consider the insertion of a tour anywhere within another tour.

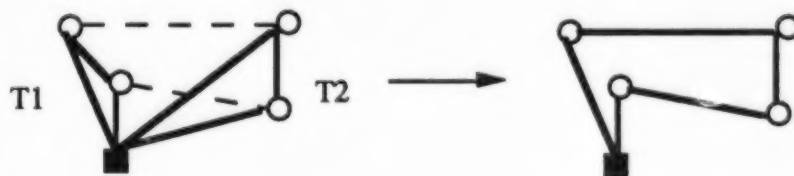
Section 4. A Generalized Clarke and Wright algorithm

The classical Clarke and Wright algorithm generates a solution to the TSP problem by iteratively linking the end points of two existing subtours until a single tour is produced. It uses a saving measure to guide this merging process. Alternatively, we can see the algorithm's basic merging process as one of inserting a tour between another tour's end point and the depot (see figure 2a).

It thus looks natural to enlarge this set of savings by considering the insertion of a tour between any two adjacent nodes of another tour (see figure 2b).



a) Clark and Wright : inserting T2 at the end of T1



b) Generalized Clark and Wright: inserting T2 anywhere between two nodes of T1.

Fig. 2 Clark and Wright vs Generalized Clark and Wright

Assuming that end points of tour T1 are *first* and *last*, insertion of T1 between nodes *i* and *j* in tour T2 generates two potential savings:

$$s_{\text{first,last},i,j}^{(1)} = d_{1,\text{first}} + d_{1,\text{last}} - (d_{i,\text{first}} + d_{\text{last},j} - d_{i,j})$$

$$s_{\text{first,last},i,j}^{(2)} = d_{1,\text{first}} + d_{1,\text{last}} - (d_{i,\text{last}} + d_{\text{first},j} - d_{i,j})$$

We can then replace steps 1, 2 and 3 of the Clarke and Wright algorithm (see section 2) by the single step 1':

Step 1'. Given the current set of tours, compute the generalized savings for each tour and each insertion point. Form a larger tour by inserting the appropriate tour at the appropriate insertion point, based on the maximal possible saving. Repeat until a single tour is formed.

This approach considers a larger set of modifications to the current solution (larger neighborhood) than the classical approach, at

the expense of more computations. Selecting a tour and an insertion point based on this generalized saving measure requires $O(N^2)$ operations (there are $O(N)$ tours and $O(N)$ insertion points) and this step is repeated $N-2$ times. Hence, the generalized Clarke and Wright algorithm works in the order of N^3 . In contrast to the classical approach, the savings are dynamically reevaluated after each modification to the current set of tours, because the insertion points are also modified in the process. Evaluating all the generalized savings at once as in the classical approach would be too costly, since only a small fraction of them are used during problem solving (the generalized savings measure involve four different nodes, so there are $O(N^4)$ such savings).

It must be noticed that the generalized approach does not cure the "myopic" behavior of the classical algorithm, but since the set of generalized savings is larger than the set of classical savings, more alternatives are considered at each iteration.

The next section now presents and compares computational results obtained with the classical approach and the two variants introduced in sections 3 and 4.

Section 5. Computational Results

In order to compare the alternative problem solving approaches presented in this paper, we ran experiments on a Sun 3/50 workstation using a routing software written in C. We randomly generated complete euclidean networks of size $N=25, 50, 75$ and 100 . The x and y coordinates of the nodes were uniformly distributed in the square $[0,1000] \times [0,1000]$.

For each size, fifty problems were generated and solved by each method. Table I thus shows, for each network size, the means of the tour's lengths as taken over those fifty problems (CW is the classical Clarke and Wright algorithm, CW-P-0.10 is the penalty algorithm with the α parameter varying by increments of 0.10 - see below -, CW-P-0.25 is the penalty algorithm with the α parameter varying by increments of 0.25 and CW-G is the generalized algorithm). Computation times are shown in seconds under each result.

It must be noticed that the penalty method was applied on each problem with two distinct sets of penalty weights. First, we applied this approach with the α parameter varying from 0 to 1 by step of 0.1 (with the special case $\alpha=1, \beta=0$ being the classical Clarke and Wright

algorithm). The penalty algorithm was thus applied 11 times, each time with a different α value, and the results shown in Table I are those obtained with the best α value on each problem. We then applied the penalty algorithm with the α parameter varying from 0 to 1 by step of 0.25. The penalty algorithm was thus applied here with only five different values.

The idea of using many different α values is to introduce "intelligent" perturbations into the Clarke and Wright algorithm by putting more or less weight on the penalty factor (pre-experimentations demonstrated that applying the penalty algorithm with a single α value generates solutions that compare to the classical approach). By enlarging the number of α values, computation times are increased but better solutions are expected to come out. Of course, the computation times are multiplied by a constant factor (i.e. the number of α values) which is independent of N .

A close examination of the results also showed that no particular values for the α parameter looked better than the others (in particular, the most "natural" value $\alpha=0.5$, which gives equal weight to the saving and the penalty factor, did not come out as better than the other values). As a matter of fact, the relative magnitude of the saving and penalty values can vary a lot from one pair of nodes to the other, and the linear combination should ideally be adapted to each saving. Reapplying the penalty algorithm with various α values stabilize the method and avoid bad solutions that could occur with a single value.

In Table I, we can see that both CW-G and CW-P generate better solutions than the classical algorithm. It is also interesting to notice that CW-P performed quite well as compared to the generalized approach, even if its complexity remains in the order of $N^2 \lg N$. CW-P-0.25 is slightly worse than CW-P-0.10, but takes about half the time to generate a solution. Hence, a trade-off must be made between the number of penalty weights to be used (which is directly related to the computation time) and the quality of the solution.

Table II shows, for each variant, the percentage of improvement over the CW tours (i.e. $\text{Length}_{\text{CW}} - \text{Length}_{\text{variant}} / \text{Length}_{\text{CW}}$). These results are quite interesting, because the classical Clarke and Wright algorithm already generates high quality solutions.

	N=25	N=50	N=75	N=100
CW	4409.5 (0.9 sec)	6173.8 (4.7 sec)	7407.7 (10.9 sec)	8407.0 (20.0 sec)
CW-P-0.25	4341.3 (4.5 sec)	6033.6 (19.5 sec)	7236.5 (43.1 sec)	8255.6 (84.6 sec)
CW-P-0.10	4327.5 (10.6 sec)	6019.9 (43.2 sec)	7199.5 (91.2 sec)	8222.9 (178.3 sec)
CW-G	4306.6 (9.2 sec)	5990.1 (78.0 sec)	7175.4 (255.0 sec)	8171.7 (585.0 sec)

Table I Computational Results

	N=25	N=50	N=75	N=100
CW-P-0.25	1.6%	2.3%	2.3%	1.8%
CW-P-0.10	1.8%	2.5%	2.8%	2.2%
CW-G	2.3 %	3.0 %	3.1 %	2.8 %

Table II % of improvement over the classical approach

Section 6. Concluding remarks

In this paper, we described two variants of the classical Clarke and Wright algorithm. Results show that the penalty algorithm is certainly an interesting alternative to the classical algorithm, since its complexity remains in the order of $N^2 \lg N$ and it systematically produces better solutions than the classical approach. However, in order to get better results, the penalty algorithm must be applied with different penalty weights. Obviously, by increasing the number of penalty weights, the computation times are also increased, as well as the improvement over the Clarke and Wright tour.

Acknowledgements. I would like to thank professors Guy Lapalme and Jean-Marc Rousseau for their useful comments and the Natural Sciences and Engineering Council of Canada (NSERC) for their financial support.

References

- [1] Clarke G., Wright J., "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", *Ops. Res.* 12, pp. 568-581 (1964).
- [2] Kruskal J., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proc. Am. Math. Soc.* 2, pp. 48-50 (1956).
- [3] Tillman F., Cain T., "An Upper Bounding Algorithm for the Single and Multiple Terminal Delivery Problem", *Management Sci.* 18, pp. 664-682 (1972).
- [4] Williams J., "Algorithm 232: HeapSort", *Commun. ACM* 7, pp. 347-348 (1974).